Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

Pattern Analysis for Autonomous Vehicles with the Region- and Feature-based Neural Network: Global Self-Localization and Traffic Sign Recognition

Jason A. Janét, Mark W. White, Troy A. Chase, Ren C. Luo and John C. Sutton, III
Center for Robotics and Intelligent Machines
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7911

Abstract: Autonomous vehicles require that all processes be efficient in time, complexity and data storage. In fact, an ideal system employs multi-functional models where ever possible. This paper presents the Region- and Feature-based neural network (RFNN) as a viable pattern analysis process engine for solving a variety of problems with a single math model. The RFNN employs receptive fields and weight sharing which compensate for noise, minor phase shifts and occlusions. The RFNN also utilizes greedy adaptive learning rates and mature feature preservation to expedite the overall training process. A novel ad hoc approach called "shocking" is used to solve the instability problem inherent to greedy adaptive learning rates. The basic RFNN "feature" is grounded in computer vision morphology in that the neural network autonomously learns subpatterns unique to various problems. This paper comprehensively describes the flexible RFNN architecture and training process and presents two problems that can be solved by the RFNN: sensor pattern-recognition and traffic sign recognition.

1. Introduction

Pattern analysis is a crucial tool for autonomous mobile robots. The ability to recognize natural landmarks from various sensors can enhance a robot's position and orientation filtering strategy. The ability to recognize a face or voice can tell a robot who is nearby. Speech and sign-language recognition enable a robot to interpret instructions given to it through a natural medium. Character recognition enables a mobile robot to *interpret* signs (instead of memorizing the entire sign). A variety of distinctly different, dedicated models exist that solve these problems individually. But if each problem required its own dedicated math model, a robot would eventually accumulate large numbers of different models, slowing access and execution. A processing engine common to all of these problems, on the other hand, would expedite pattern analysis, simplify execution and reduce storage requirements. It is believed that the Region- and Feature-based Neural Network (RFNN) provides such a processing engine that is easy to implement, compact and fast.

In this paper we describe in detail the flexible RFNN architecture and training. While slightly involved, the RFNN notation presented here permits the construction of fully- and partially-connected neural networks with and without receptive fields and weight sharing. This paper also presents a method called *shocking* that stabilizes the training process when greedy adaptive learning rates are used. Shocking, hence, expands the range of problems that can be solved using back-propagation. Finally, this paper presents an example of how preserving mature feature synaptic weights expedites the training process and enables architectures to be pruned and/or enlarged without a severe redesign penalty.

While there are numerous examples of complex problems solved by the RFNN, space limitations require that we only present two: 1) sensor pattern-recognition for autonomous mobile robot global self-localization (GSL) [11, 12]; and 2) outdoor traffic sign-recognition. Other areas of pattern analysis we are currently investigating include: 1) sensor pattern recognition for topological cues [16]; 2) face recognition; 3) landmark recognition; and 4) optical character recognition [15].

1.1. The Region- and Feature-based Neural Network

The RFNN is a self-organizing multi-functional neural network architecture that embodies many new paradigms in the artificial neural networks field which expedite training and recall, minimize computational complexity and data size as well as accommodate noisy, incomplete and/or phase shifted data. The RFNN is also designed under the premise that a neural network should be able to retain all it has previously learned about various features even if the architecture is enlarged or pruned.

1.2. The RFNN for Pattern Analysis

A common argument against the use of multi-layered neural networks in pattern analysis is that the architecture constrains on-line adaptation. That is, new patterns (presumably) cannot be learned without retraining with the entire training set since there is usually no provision for an increase in the number of classes without a severe redesign penalty. It is for this reason that many believe multi-layered neural networks do not offer any significant advantage over dedicated models.

However, use of the RFNN has shown that it *is* possible to not only *add* new classes (additional output neurons, new synaptic connections and more hidden layer neurons) but to also *prune* seemingly irrelevant (output neurons, synaptic connections and hidden layer neurons). In fact, it has been found that, while a new class requires exposure to other known classes (a very human-like requirement and, hence, reasonable provision), the RFNN architecture does **not** incur a severe redesign penalty especially if it can utilize previously learned features. The new class (new output neuron) merely decides which previously learned features are relevant. If these previously learned features are *locked*, only hidden to output synaptic weights need to be changed in the training

0-7803-2988-4/96 $4.00 © 1996 IEEE

process. The next section defines "regions" and "features" and describes how the RFNN architecture can accommodate new and, for that matter, removed classes. Section 3 describes the training process. Section 4 discusses how the RFNN can solve the GSL problem. Section 5 shows how the RFNN can use previously learned features trained on a subset of classes to expedite training when new classes are added.

2. RFNN Architecture

The RFNN software provides a flexible, multi-layered, feed-forward neural network as well as the ability to add to and prune from the architecture even after training has begun. The user simply specifies the desired architecture, initialization parameters, learning parameters and stop-training conditions [15]. Figure 1 shows a typical multi-layered, feed-forward neural network that is ideal for back-prop learning.

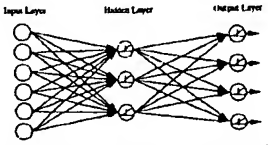

Figure 1. Multi-layered, feed-forward architecture.

2.1 Regions: Input-to-Hidden Neuron Connectivity

The input layer is where training and testing data is introduced to the neural network. The problem space can be broken down into two-dimensional *regions*. Regions can overlap each other or define independent portions of the problem space. Figure 2 shows an example problem space with two overlapping regions.

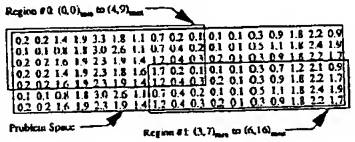

Figure 2. Example problem space with two overlapping regions.

The arrangement of regions defines the connectivity between the input layer and hidden layer neurons. Specifically, each region has its own dedicated hidden layer. If the desired architecture specifies fully connected input-to-hidden neurons, there would be only one region. Otherwise, the input-to-hidden neurons can be made partially connected by breaking the problem space into more than one region. For receptive field architectures, regions define the sections of the problem space inside which feature kernels can search for pattern matches.

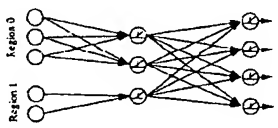

Figure 3. Regions define input-to-hidden neuron connectivity.

2.2 Features: Hidden Neuron Convolution Kernels

We use the term *feature* because the RFNN bases its final output on the quantity and/or location of certain characteristic features unique to an overall larger class of objects, images or patterns. Some features might be shared by several classes (like the color red, for example). But each unique class is characterized by the degree and locations of particular features that are found in the region.

A *feature* is a movable hidden layer neuron with a unique set of input-to-hidden layer synapses. The number of hidden layer neurons for a given region is dictated by the number of features it has. The set of input-to-hidden layer synapses (receptive field) unique to a feature is held in a *feature weight matrix* (FWM). See figure 4.

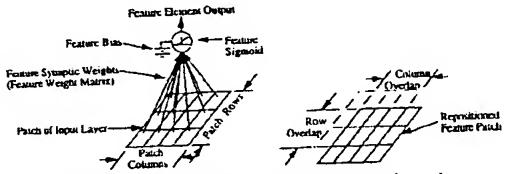

Figure 4. Feature neuron, FWM, patch and overlap.

The FWM is convolved with its region to search for a possible subpattern match. This convolution process is similar to morphological operations in computer vision where a kernel is compared to parts of an image to look for a subpattern match [7]. The number of input-to-hidden layer synapses in each FWM is defined by the feature *patch* size (kernel size). Specifically, a patch is a subset of its respective region and, hence, its size is the number of rows and columns ($\leq$ region's number of rows and columns) inside the problem space.

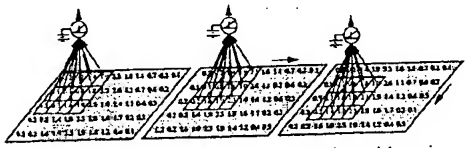

Figure 5. Feature patch, neuron convolving with region.

2.3 Feature Patch Size and Overlap

To determine if, where and to what degree a feature occurs, the feature patch is placed over portions of its respective region. This operation is like sliding a window over a region of the problem space where, for each position of the window, the portion of data inside that window can be measured by the feature neuron for similarity to a relevant pattern. Placement of each feature patch depends on the *overlap* parameters. Specifically, the overlap size defines the amount of rows and columns a patch can be overlapped when it is repositioned. Patch overlapping can compensate for slight feature rotations and translations by searching areas of the region *at* and *near* the location of the expected occurrence. See figure 5.

In general, we want to minimize the number of

features to minimize the overall architecture data size. Choosing the optimal number of features for each region can seem somewhat subjective and arbitrary. One might think, for example, that the RFNN will need to look for horizontal and vertical lines in an image and, thus, decide that only two features are necessary. In truth, however, this way of envisioning how the RFNN learns a feature might be completely different from how it *actually* learns a feature. After all, unlike computer vision morphology, the neural network decides for itself how a feature should be measured. This is because the RFNN not only looks at *whether* a candidate feature does or does not completely match; it also looks at the *degree* of match. This view of RFNN training and recall is similar to the factorial-code based approach to face-recognition proposed by Attick [1].

2.4 Hidden-to-Output Connectivity with the RFNN

For each position of a feature inside its region, a unique value associated with that position is produced by the feature neuron. These position-dependent feature values are placed in a *feature element matrix* (FEM) where they are held and later presented to the output neuron layer. Every feature has its own FEM. The general idea behind the FEM is to let the output layer know where and to what degree certain features were found. Essentially, this creates the illusion that there are many more hidden neurons in the architecture since there is a hidden neuron output value for each position (FEM element) of a feature patch. In truth, however, this is not the case since a single feature will use the same input-to-hidden synaptic weights at each patch placement. Hence, synapses exist between the FEM elements and the output neurons. Figure 6 shows how a FEM is filled and connected to an output neuron. The set of FEM element-to-output weights is different for each output neuron. This is because each output neuron is entitled to place as much or as little significance on a given type of feature. After all, not all feature types are expected to be relevant to all output neurons.

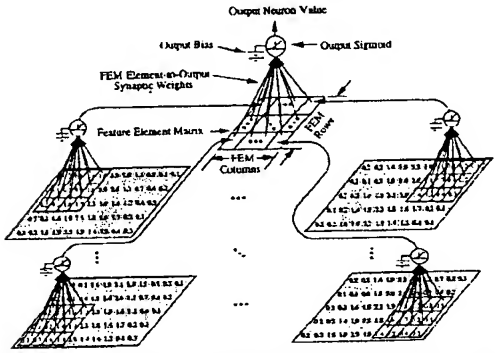

Figure 6. Connectivity of a FEM's elements to an output neuron.

3. RFNN Training with Backprop

Due to the uniqueness of architectures using receptive-fields and weight sharing, the back-propagation training algorithm needs to be adapted to the RFNN. For an architecture with $r = 1,2...R$ regions, the r^{th} region is defined in the problem space W by $r \in \left[\left(row^r_{\min}, col^r_{\min}\right), \left(row^r_{\max}, col^r_{\max}\right)\right]_W$. For each region r there are $f = 1,2...F_r$ features each of patch size $P^r_{size} = \left(N^{P^r}_{rows}, N^{P^r}_{cols}\right)$ and overlap $O^r_{size} = \left(N^{O^r}_{rows}, N^{O^r}_{cols}\right)$. Hence, for each feature f the FEM size is $\underline{FEM}^{r,f}_{size} = \left(N^{FEM^{r,f}}_{rows}, N^{FEM^{r,f}}_{cols}\right)$ where,

$$N^{FEM^{r,f}}_{rows} = \left\lfloor \frac{\left(row^r_{\max} - row^r_{\min}\right)_W - N^{O^r}_{rows} + 1}{\left(N^{P^r}_{rows} - N^{O^r}_{rows}\right)} \right\rfloor,$$

$$N^{FEM^{r,f}}_{cols} = \left\lfloor \frac{\left(col^r_{\max} - col^r_{\min}\right)_W - N^{O^r}_{cols} + 1}{\left(N^{P^r}_{cols} - N^{O^r}_{cols}\right)} \right\rfloor \quad (1)$$

and the FWM of input-to-hidden synaptic weights is

$$\underline{FWM}^{r,f} = \begin{vmatrix} \omega^{r,f}_{0,0} & \cdots & \omega^{r,f}_{0,N^{P^r}_{cols}} \\ \vdots & \ddots & \vdots \\ \omega^{r,f}_{N^{P^r}_{rows},0} & \cdots & \omega^{r,f}_{N^{P^r}_{rows},N^{P^r}_{cols}} \end{vmatrix}$$ where each

synaptic weight, $\omega^{r,f}_{m,n}$ corresponds to the m^{th} row and n^{th} column of the FWM for feature f in region r. Also, each feature has a bias weight associated with it, $\omega^{r,f}_b$.

Each cell of $\underline{FEM}^{r,f}$ is filled by the sigmoid

$$f\left(net_{i,j}\right) = \frac{1}{\left(1 + e^{-net_{i,j}}\right)} \quad (2)$$

where i and j are indices of $\underline{FEM}^{r,f}$ and

$$net_{i,j} = \sum_{m}^{N^{P^r}_{rows}} \sum_{n}^{N^{P^r}_{cols}} \left(\omega^{r,f}_{m,n} a_{x,y}\right) + \omega^{r,f}_b. \quad (3)$$

The term $a_{x,y}$ is the value of an input element from the problem space inside region r. The indices x and y represent the row and column, respectively, of the input element in the problem space reference frame W. Thus,

$$x = \left[row^r_{\min} + (i)\left(N^{P^r}_{rows} - N^{O^r}_{rows}\right) + m\right],$$
$$y = \left[col^r_{\min} + (j)\left(N^{P^r}_{cols} - N^{O^r}_{cols}\right) + n\right] \quad (4)$$

If, as is the case with the RFNN, the neural network is fully connected between the FEM elements and output neurons, there exists a synaptic weight between every output neuron and every cell of every FEM. This gives the RFNN the tendency to be top-heavy. That is, since each output neuron considers the value associated with each placement of the feature patch, there can be significantly more FEM element-to-output neuron synapses than input-to-feature neuron synapses. So each output neuron $o = 1,2,...O$ assumes a value according to

$$f(net_o) = \frac{1}{\left(1 + e^{-net_o}\right)} \tag{5}$$

where,

$$net_o = \sum_r^R \sum_f^{F_r} \sum_i^{N_{rows}^{FEM^{r,f}}} \sum_j^{N_{cols}^{FEM^{r,f}}} \left[\omega_{r,f,i,j}^o FEM(i,j)\right] + \omega_b^o. \tag{6}$$

Although this notation is more involved than simpler forms of fully connected feed-forward neural networks [8, 9], it allows us to construct architectures with receptive fields and weight sharing. It should also be noted that, if an architecture should not need receptive fields, the RFNN patch should be the same size as the region (FEM matrices are one element in size) and that the number of hidden layer neurons directly corresponds to the number of features. Hence, this notation provides greater flexibility by giving the option of specifying receptive fields for the architecture.

3.1 Error Back-Propagation

The back-propagation algorithm for the RFNN is also an extension of the standard back-prop algorithm. That is, special learning provisions must be made for the movable feature patch. Since feature patches are not necessarily fixed over the problem space, each individual synapse can be affected by a multitude of input values per data example $q = 1,2,\ldots Q$. From [8] the task of the network is to learn associations between a set of input-output pairs $\left\{\left(\underline{p}_1, \underline{q}_1\right), \left(\underline{p}_2, \underline{q}_2\right), \ldots \left(\underline{p}_Q, \underline{q}_Q\right)\right\}$. The performance index for the network is

$$V = \frac{1}{2}\sum_q^Q \left(\underline{t}_q - \underline{o}_q\right)^T \left(\underline{t}_q - \underline{o}_q\right) = \frac{1}{2}\sum_q^Q \underline{e}_q{}^T \underline{e}_q \tag{7}$$

where $\underline{o}_q$ is the vector of output neuron values resulting from the q^{th} input, $\underline{p}_q$, and $\underline{e}_q = \underline{t}_q - \underline{o}_q$ is the error for the q^{th} input. Using steepest descent, the performance index for a single input/output pair can be approximated by $\hat{V}_q = \frac{1}{2}\underline{e}_q{}^T \underline{e}_q$. For epoch $k = 1,2,\ldots K$ the approximate steepest (gradient) descent algorithm for the FEM element-to-output neuron synapse and output neuron bias weights require that

$$\omega_{r,f,i,j}^o(k+1) = \omega_{r,f,i,j}^o(k) - \alpha_{r,f,i,j}^o(k)\sum_q^Q \frac{\partial \hat{V}_q(k)}{\partial \omega_{r,f,i,j}^o(k)} \tag{8}$$

and

$$\omega_b^o(k+1) = \omega_b^o(k) - \alpha_b^o(k)\sum_q^Q \frac{\partial \hat{V}_q(k)}{\partial \omega_b^o(k)} \tag{9}$$

where $\alpha_{r,f,i,j}^{o,k}$ and $\alpha_b^{o,k}$ are the adaptive learning rates at epoch k for the individual FEM element-to-output neuron synapses and output bias synapses respectively. Similarly, the steepest descent algorithm for feature biases requires

$$\omega_b^{r,f}(k+1) = \omega_b^{r,f}(k) - \alpha_b^{r,f}(k)\sum_q^Q \frac{\partial \hat{V}_q(k)}{\partial \omega_b^{r,f}(k)}. \tag{10}$$

Unlike the standard synaptic weight update rules of (8)-(10), the feature patch synaptic weights (FWM) must account for every placement in the region. Hence,

$$\omega_{m,n}^{r,f}(k+1) = \omega_{m,n}^{r,f}(k) - \alpha_{m,n}^{r,f}(k)\sum_q^Q \sum_i^{N_{rows}^{FEM^{r,f}}} \sum_j^{N_{cols}^{FEM^{r,f}}} \frac{\partial \hat{V}_q(k)}{\partial \omega_{m,n}^{r,f,i,j}(k)}. \tag{11}$$

3.2 Greedy Adaptive Learning Rates

It has been shown that "the method of adaptive learning rates [10] is much faster than steepest descent, generally reducing training time by an order of magnitude, and it is also very dependable. It is not prone to get into trouble and does not require special care...[It] is fast, dependable, and highly automatic..." [20, 24]. What follows is a greedy version of the adaptive learning rates method that has been tailored to the RFNN. In general, each synaptic weight in a neural network architecture can have its own learning rate. We are concerned with the *direction* in which errors for a synaptic weight decrease over an exponential average f_ω,

$$f_\omega(k+1) = \theta \cdot f_\omega(k) + (1-\theta) d_\omega(k). \tag{12}$$

$d_\omega(k) = \sum_q^Q \frac{\partial \hat{V}_q(k)}{\partial \omega(k)}$ and θ defines the weighting of consecutive error associations $(\theta = 0.7)$. The signs of d_ω and f_ω give us a precise measure of the direction in which the error decreases both currently and recently. The equation for the changing value of the learning rate for a synaptic weight, α_ω, is

$$\alpha_\omega(k+1) = \begin{cases} \kappa\alpha_\omega(k) & \langle d_\omega(k) f_\omega(k) > 0,\ \alpha_\omega(k) < \tau \rangle \\ \phi\alpha_\omega(k) & \langle d_\omega(k) f_\omega(k) \le 0 \rangle \end{cases} \tag{13}$$

where typically $(\kappa = 1.1)$ and $(\phi = 0.5)$.

3.2.1 Adaptive Learning Rate Heuristic Modification

One extremely important difference between (13) and the heuristic model in [24] should be noted; instead of *adding* $(\kappa = 0.1)$ to the learning rate, we *multiply* $(\kappa = 1.1)$. The need to do this stems from the fact that architectures with many synapses can saturate [9] if their learning rate is too high. In fact, extremely large architectures might have initial learning rates on the order of $\alpha_\omega(0) = 0.001$. Hence, adding $(\kappa = 0.1)$ to α_ω could have too radical an effect on the backpropagation process. Instead, a subtle 10% increase of the learning rate each epoch reduces the tendency toward saturation regardless of the initial learning rate magnitude.

Another very important detail from (13) not mentioned in [24] is that an upper bound τ must be placed on α_ω. Typically, $(\tau \le 1)$ since it is commonly desired that ω not change radically based on the back-propagated

error (particularly in later stages of training). However, in the spirit of proportional- and derivative-controllers, learning rates can anticipate (i.e., become *greedy*) if $(\tau > 1)$. The expected result is an even faster convergence to the global minimum. But with this greed comes the risk of saturation. That is, excessively large learning rates can make synaptic weights overshoot the global minimum to a point on the error surface from which the neural network can sometimes not recover.

3.2.2 Stabilization with Adaptive Learning Rate Shocking

Our research has shown, however, that it is possible to let $(\tau > 1)$ and still maintain relatively stable convergence with an *ad hoc* approach called *shocking*. In fact, with the XOR problem alone, shocking reduced the number of failed convergences by as much as 33% [15]. Simply stated, *shocking* a neural network involves reducing all synaptic learning rates to a small value (near their initial value). The conditions for shocking are quite simple and, yet, they can significantly reduce failure rates. An in-depth analysis of the effects of shocking has shown that it significantly reduces the chances of saturation and, hence, stabilizes greedy adaptive learning [15].

The *first heuristic* stipulates that if the error at epoch $k+1$ of a neural network in training increases to a value larger than the error at epoch k, the learning rates should be shocked. Temporarily reverting to small learning rates presumably gives the RFNN the opportunity to quickly (re)turn to its original destination or, due to the instability that triggered the shock, locate a better minimum on the error surface. The *second heuristic* condition for shocking requires that if the learning rates are large enough to significantly impact training, but the training error is decreasing at a very slow rate, the learning rates should be shocked. The reason for this is as follows: It is believed that when learning rates are very large, it is possible for the synaptic weights to overshoot and vacillate over global minima while still (very) slowly converging and not causing the neural net to activate the first heuristic.

4. Sensor Pattern Recognition with the RFNN

An example application where the RFNN is used to recognize sensor patterns involves enabling a robot to determine which topographical region it is in [16]. To date significant work has been devoted to developing self-localization approaches for autonomous mobile robots that depend on prior dead reckoning estimates and discrete-time models to rationalize and correct robot position and orientation based on correlations between predicted and actual sensor data [5, 6, 13, 14, 19, 22]. But, without an *initial* reliable position estimate, most become ineffective.

Our objective is to be able to determine which region of space the robot is in without knowing how it got there. Specifically, we treat the global self-localization (GSL) of a mobile robot as an optical character recognition (OCR) problem [11]. That is, natural landmarks can, through computer vision and/or mapped sonar data, assume char-

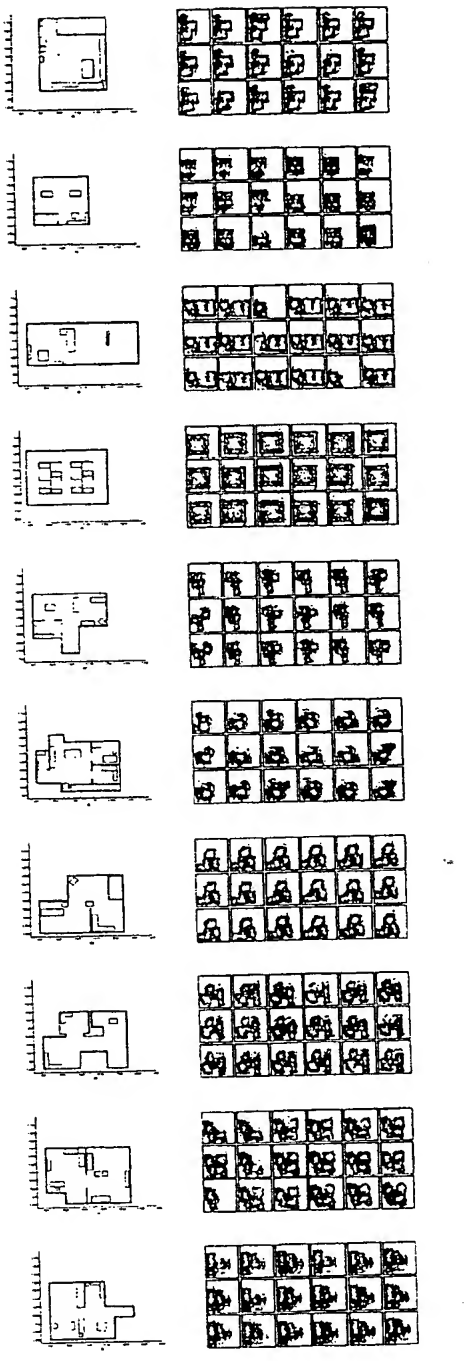

Figure 7. Geometric rooms and corresponding digitized images.

acter-like configurations unique to those landmarks. Our research has approached the GSL problem using mapped simulated sonar data [12] with great success.

Figure 7 shows digitized forms of mapped simulated sonar data (gathered while exploring the room) used by the robot to determine which room it is in. The problem space is a 29x29 binary pixel grid covering an area 16 meters wide and 12.7 meters long. A perceived echo inside a pixel's boundaries activates that pixel. Preprocessing for GSL is limited to gross shifting (moving the minimum x- and y-coordinates of the echo points to a common global reference frame) and digitization. Otherwise, with the help of receptive fields, the GSL problem is time-, translation- and rotation-invariant. Architecture sizes are minimized (four or fewer 5x5 features). Classification is 96.7% to 98.9% for the 180 test data sets covering a full range of odometric and sensor noise. Also, the OCR-like architecture sizes of the RFNN are actually 10% to 25% that prescribed by LeCun and Sackinger [18, 23].

5. Traffic Sign Recognition with the RFNN

A good test of a an autonomous vehicle's ability to recognize landmarks can be found in the problem of identifying traffic-signs in natural scene images. A four hidden layer neural network using receptive fields like that presented in [17] was shown to be 99% successful at recognizing nine traffic signs [21]. However, as mentioned in Section 4 this type of neural network can be extremely slow due to its large size and, thus, complexity.

An even slower approach based entirely on color and form was shown by [4] to successfully detect sign type at a fixed distance from the camera. Another approach that detects sign type using contours generated by joining image edges is shown in [3] although, here again, the actual contents of the sign are not identified. Finally, an approach using an Incremental Fractal Brownian Motion (IFBM) model and recurrent self-organizing Kohonen neural networks was implemented to successfully identify landmark signs in natural scene images in an isotropic and statistically stationary manner [2]. Despite its success, the IFBN approach required a substantial amount of preprocessing and the Kohonen neural network required an even greater amount of time to train.

5.1 Approach

Our primary intention is to see if the RFNN can use only a few signs to quickly learn significant features that can be used to classify other signs. Of course, we must also assess the impact on training time and architectural modifications. That is, we must compare the training time for a neural net that uses previously learned features with the training time for a neural net that trains entirely on the full set of training data.

Using an in-house traffic sign data base of 92 natural scene images, we were able to test the capabilities of the RFNN. The images in the database are color photographs of individual and sets of traffic signs taken over the span of two months under varying light/weather conditions, zooms and pose angles. The photographs were taken in the following east-coast states: North Carolina, Virginia, West Virginia and Pennsylvania. The RFNN used gray scale versions of the color images. A SunSPARClassic was used to train and test the RFNN.

To compensate for zoom, the segmented sign image is fit to a problem space of 50 rows by 50 columns. To compensate for light intensity variations, each sign segmentation undergoes a histogram equalization. These are the only two steps in the pre-processing stage of recognition. That is, gray-scale image data is presented to the RFNN through the problem space. The entire problem space constitutes the region of interest. Hence, the traffic sign recognition architectures have only one region.

5.2 Using Previously Learned Features

Figure 8 shows 32 sample images from the 92 sign data base that was used to train and test the RFNN.

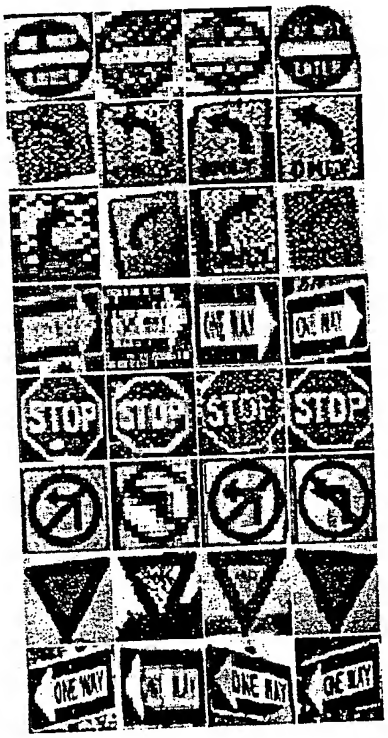

Figure 8. Sample set of traffic sign images for the RFNN.

We want to quickly extract a set of features that can be used to identify a broad spectrum of traffic signs like those in Figure 8. At a glance, the three sign groups appear to have features in common: 1) *DO NOT ENTER* and *STOP* both have similar shapes, shades and a cluster of white in the center. The *YIELD* sign, while of a

different shape, consists of similar shades; 2) The *ONE WAY* signs have similar features for both directions (right and left); 3) The *LEFT TURN ONLY*, *RIGHT TURN ONLY* and even the *NO LEFT TURN* and *NO RIGHT TURN* have similar features in that a dark arrow is in the (near) center of each predominantly white sign.

Figure 9. Basic set of traffic signs for feature extraction.

To train four features on the three signs in figure 9, a single region, 4 feature, 3 output RFNN had a training time of 1:32:45 (hours:minutes:seconds). With these features, a single region, 4 feature, 8 output neural net was then trained on a 50 example training set. The patch size was 10x10 and the overlap was maximized at 9x9. The normalized per epoch training time was 0.34375 sec/epoch/example/output neuron. Total training time was 14:17:22 (including time spent training the four features).

For the neural network that did not use previously learned features, the normalized training time was 2.2335 sec/epoch/example/output neuron. And the total training time was 80:37:05. Both of these numbers are between six and seven times larger than the training times for the neural network that used previously learned features. Table 1 shows this comparison.

Table 1. Training times for neural networks with and without previously learned features.

Training Model	Feature Training Time (hr:min:sec)	Normalized per-epoch Training Time	Total Training Time (hr:min:sec)
Without Previously Learned Features	N/A	2.2335 sec	80:37:05
With Previously Learned Features	1:32:45	0.34375 sec	14:17:22

5.3 Classification

Classification was based on the highest output neuron value in excess of 0.8. The RFNN correctly classified all 92 signs at a recall rate of 0.2 seconds per image. From looking at the images in figure 8, it is clear that the RFNN could correctly tolerate rotations up to about 25°, variations in light conditions and large variations in zoom. It should also be noted that if the RFNN could correctly classify these signs using gray scale, it would probably be even more robust using color images.

6. General Comments

The most significant impact of this research is the consolidation of pattern analysis math models into one problem solving engine, the RFNN. The math presented in this paper enables the construction of neural networks with simple architectures and receptive fields and weight sharing. The adaptive learning rate modification presented in this paper can be used to reduce the risk of saturation in training. The shocking heuristics further reduce the risk of saturation (failed convergence) with a simple *ad hoc* solution. The research discussed in this paper suggests that the RFNN is capable of solving at least the sensor-pattern and traffic-sign recognition problem with minimal training time, recall time, data storage and, especially, pre-processing. The traffic-sign recognition problem, in particular, demonstrates how a neural network can utilize previously learned features (from a subset of classes) to expedite the training process when new classes are added.

Bibliography

[1] J. J. Atick, P. A. Griffin and A. N. Redlich. "Face Recognition from Live Video for Real-World Applications-Now." *Advanced Imaging*, May 1995.

[2] M. Azam, H. Potlapalli, J. A. Janét and R. C. Luo. "Outdoor Landmark Recognition Using Segmentation, Fractal Model and Neural Networks" *Proc. ARPA Image Understanding Workshop*, Feb. 1996

[3] B. Bessete, S. Estable, B. Ulmer, D. Reichardt. "Shape Classification for Traffic Sign Recognition." *First IFAC Int'l Workshop on Intelligent Autonomous Vehicles*, UK, 1993.

[4] M. Blancard, "Road Sign Recognition: A Study of Vision-Based Decision Making for Road Environment Recognition." *Vision Based Vehicle Guidance*, Ichiro Masaki, editor, Springer-Verlag, 1992, pp. 162-175.

[5] I. J. Cox and J. J. Leonard. "Probabilistic Data Association for Dynamic World Modeling: A Multiple Hypothesis Approach." in *Fifth Int'l Conf. on Adv. Robotics*, 1991.

[6] H. R. Everett, et al. (1990). Modeling the Environment of a Mobile Security Robot. *Tech. Doc. 1835*. San Diego, CA, Naval Ocean Sys Ctr.

[7] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision: Volume 1*. Addison-Wesley Publishing Co., Reading, MA, 1992.

[8] M. T. Hagan and M. B. Menhaj. "Training Feedforward Networks with the Marquardt Algorithm." *IEEE Trans. on Neural Networks*, Vol 5(6), November, 1994.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Co., NY, 1994.

[10] R. A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks* Vol. 1(4). pp 295-308, 1988.

[11] J. A. Janét, R. Gutierrez, T. A. Chase, M. White and R. C. Luo, "Global Self-Localization for Autonomous Mobile Robots Using Kohonen Neural Networks." *IEEE/RSJ Int'l Conference on Intelligent Robotics and Systems*, Pittsburgh, PA, August, 1995.

[12] J. A. Janét, R. Gutierrez, T. A. Chase, M. White and R. C. Luo, "Global Self-Localization for Autonomous Mobile Robots Using Region- and Feature-based Neural Networks." *IEEE/SICE/AEI Int'l Conf. on Industrial Electronics*, Nov. 1995, Orlando, FL.

[13] J. A. Janét, R. Gutierrez-Osuna, R. C. Luo, "Autonomous Mobile Robot Self-Referencing with Sensor Windows and Neural Networks." *IEEE/SICE/AEI International Conference on Industrial Electronics*, Nov. 1995, Orlando, FL.

[14] J. A. Janét, R. C. Luo, M. G. Kay. "Autonomous Mobile Robot Global Motion Planning and Geometric Beacon Collection Using Traversability Vectors." *accepted to IEEE Transactions on Robotics and Automation*, 1995.

[15] J. A. Janét. *The Region- and Feature-based Neural Network Software: A User's Guide*. North Carolina State University, 1996.

[16] D. Kortenkamp and Terry Weymouth, "Topological mapping for mobile robots using a combination of sonar and vision sensing" *AAAI '94*.

[17] Y. LeCun, B. Boser, et. al. "Backpropagation Applied to Handwritten Zip Code Recognition." *Neural Computation*, Vol 1, 1989, pp 541-551.

[18] Y. LeCun, B. Boser, et. al. "Handwritten digit recognition with a back-propagation network." *Advances in Neural Information Processing Systems 2*, (D. S. Touretsky, ed.), San Mateo, CA: Morgan Kaufmann, 1990.

[19] J. Leonard and H. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publishers. Norwell, Massachusetts. 1992.

[20] R. P. Lippman, "An Introduction to Computing with Neural Nets." *IEEE ASSP Magazine*, April 1987, pp. 4-22.

[21] R. C. Luo, Harsh Potlapalli, and D. E. Hislop, "Translation and Shift Invariant Landmark Recognition Using Receptive Field Neural Networks", *1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, North Carolina, July, 1992.

[22] J. Manyika and H. Durrant-Whyte. *Data Fusion and Sensor Management: A decentralized Information-Theoretic Approach*. Ellis Horwood, NY, 1994.

[23] E. Säckinger, B.E. Boser, J. Bromley, Y. LeCun and L.D. Jackel, "Application of the ANNA neural network chip to high-speed character recognition." *IEEE Transactions on Neural Networks*, pp 498-505, 1992.

[24] M. Smith. *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, NY, 1993.